Computer Architecture PA4

Write-Up

2019-11563

자유전공학부

김지원

The function starts by calculating index and tag for each L1 and L2 cache.

After calculation of each index and tag, it checks for L1 hit first. With the for loop it iterates through the lines in set which is cache[index\_l1]. If L1 is hit, it increments and updates the line’s lru counter and also changes its dirty bit to 1 if the operation is ‘write’. If L1 hit, it immediately returns.

If there’s no hit, it continues in the ‘L1 miss’ case. The for loop then iterates through the set space to see if there’s any empty slots open. If there is one, it puts the currently accessed data into this cache.

If there’s no space in L1 left (l1\_filled ! = 1), eviction from L1 is needed. In order to do this, first you should get the evict index (which is ‘min’ here) according to the eviction policy. After getting the target line, if checks if that cache’s dirty bit is 1. If so, it is ‘L2\_Write’ and it updates the same data’s dirty bit in L2 cache to 1. Also, after eviction, updating the cache information is followed.

After finishing operation in L1 cache, it proceeds to do the same operations in L2 cache as well. First check if ‘L2\_Hit’. If so, return. If not, it proceeds to look for a blank space in L2 cache. The rest of the operation here is very similar to what we did in L1 cache. The only difference is, if we’re supposed to evict a data from L2 cache, and if that data’s dirty bit is 1, we need to ‘Mem\_Write’ to update that data to the memory. After that, filling in the evicted cache space with the current data being accessed is processed.